



Fehlerkorrigierende Codes

Benjamin Schmid

In vielen Situationen ist es hilfreich, Fehler bei der Datenübertragung bzw. -speicherung zu erkennen oder sogar korrigieren zu können. Hierfür gibt es eine Vielzahl an Verfahren, die verschiedene Vor- und Nachteile haben.

Inhaltsverzeichnis

1 Einführung	2
2 Barcodes	2
3 Paritätsbit	4
4 QR-Codes	6
5 Was haben wir gelernt?	8
6 Teste dich selbst	8
7 Bild-Quellen	8
8 Lösungen	9

Kapitel 1: Einführung

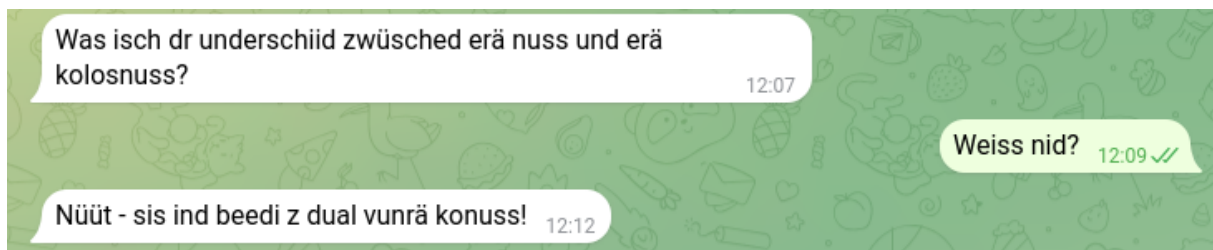


Abbildung 1: Ein Beispiel für Fehlerkorrektur bei natürlicher Sprache.

Im Beispiel in Abbildung 1 wird es dir leicht gefallen sein, den Text zu lesen, obwohl einige Fehler vorhanden sind. So handelt es sich wahrscheinlich um eine Kokosnuss, nicht um eine Kolosnuss. Und bei „sis ind“ ist wohl das Leerzeichen verrutscht und das sollte „si sind“ heissen. Es scheint uns also möglich zu sein, Fehler automatisch zu erkennen und dann auch gleich zu korrigieren.

Unsere Sprachen haben eine gewisse Redundanz, die es erlaubt, bei Fehlern die ursprüngliche Bedeutung wieder zu rekonstruieren. Wir hätten gerne, dass dies auch unsere Computer können. Doch leider kann man dies nicht einfach 1:1 übernehmen. Wir arbeiten viel mit Kontext und Erfahrung, um die Fehler zu korrigieren. Und viele Daten, die der Computer verarbeitet, haben keine eingebaute Redundanz. So ist z.B. nicht klar, wenn bei einem Auftrag zur Geldüberweisung ein Betrag von 100 CHF steht, ob dies korrekt ist oder doch nur 10 CHF sein sollte.

Wir wollen uns anschauen, wie der Computer damit umgehen kann. Zunächst versuchen wir nur zu erkennen, dass ein Fehler vorliegt. Später möchten wir aber auch den Fehler gleich korrigieren können.

Kapitel 2: Barcodes

Der erste Schritt ist, dass wir zunächst erkennen, dass ein Fehler vorliegt. Teilweise reicht dies auch schon und man benötigt keine Korrektur. Ein weit verbreitetes Beispiel dafür sind Barcodes auf Produkten, die du im Laden kaufen kannst.



Abbildung 2: Barcode eines Buches (EAN Nummer).

Beim Scannen an der Kasse möchte man sichergehen, dass der Barcode korrekt gescannt wurde. Der Kunde möchte schliesslich nicht für ein falsches Produkt bezahlen

müssen. Uns reicht aber eine Fehlererkennung, da wir den Barcode einfach nochmals scannen können oder sonst das Produkt manuell in der Kasse eintippen können.

Dies erreicht man dadurch, dass man zur eigentlichen Nummer noch eine weitere Ziffer hinzufügt, die sogenannte Prüfziffer. Im Beispiel in Abbildung 2 ist die Produktnummer 978-3-455-50236 und die Prüfziffer ist 7.

Für die Berechnung der Prüfziffer geht man wie folgt vor:

- Schreibe die Ziffern der Zahl abwechselnd auf zwei Zeilen.
- Berechne die Summe aller Ziffern (auch Quersumme genannt) für beide Zahlen.
- Multipliziere die zweite Quersumme mit 3.
- Suche eine einstellige Zahl, sodass die Summe mit den beiden Quersummen am Ende eine 0 hat. Diese Zahl ist die Prüfziffer.

Im Beispiel oben sieht dies dann wie folgt aus:

Zahl	Quersumme	Produkt
984503	$9 + 8 + 4 + 5 + 0 + 3 = 29$	$1 \cdot 29 = 29$
735526	$7 + 3 + 5 + 5 + 2 + 6 = 28$	$3 \cdot 28 = 84$

Die Summe der Produkte ergibt $29 + 84 = 113$. Zählt man 7 dazu, ergibt dies $113 + 7 = 120$, eine Zahl mit einer 0 am Ende. Somit ist 7 unsere Prüfziffer.

Der Scanner kann genau die gleiche Berechnung durchführen. Stimmt die berechnete Prüfziffer nicht mit der gelesenen Prüfziffer überein, ist ein Fehler aufgetreten.

Übung 1

Verändere eine der Ziffern in der Produktnummer und führe die Berechnung nochmals durch. Stimmt die Prüfziffer noch?

Übung 2

Berechne die Prüfziffer für die folgende Produktnummer: 401234627800

Übung 3

Beim Einlesen eines Barcodes bei der Kasse muss man primär erkennen, wenn eine Ziffer falsch gelesen wurde. Die Prüfziffer hilft aber auch bei anderen Fehlern, wenn wir die Nummer von Hand eintippen. Was für ein Fehler ist das und warum schützt die Prüfziffer davor?

Kapitel 3: Paritätsbit

Ein Computer arbeitet mit Binärzahlen. Hier sind die Ziffern entweder 0 oder 1. In vielen Fällen liegen unsere Daten daher im Binärformat vor und wir wollen ein Verfahren haben, um hier Fehler zu erkennen. Dieses Verfahren können wir dann generisch verwenden, unabhängig davon, was diese Bits darstellen.

Eine sehr einfache Art der Fehlererkennung ist das sogenannte Paritätsbit. Wir fügen jeweils ein weiteres Bit hinzu. Das zusätzliche Bit ist 1, falls wir bisher eine ungerade Anzahl an 1 hatten, sonst ist es 0. Somit haben wir schlussendlich immer eine gerade Anzahl 1.

Ein paar Beispiele:

- 11101000**0**
- 01000101**1**
- 10111001**1**
- 11100010**0**
- 11000111**1**

Übung 4

Wie viele Bits dürfen sich maximal ändern, damit wir den Fehler immer noch erkennen?

In vielen Fällen wollen wir nicht nur erkennen, dass ein Fehler vorliegt, sondern möchten diesen auch korrigieren können. Dies ist hilfreich in Fällen, in denen wir nicht auf eine Alternative zurückgreifen können.



Abbildung 3: Ein von der Witterung beschädigter QR-Code.

Im Beispiel des Barcodes konnten wir einfach nochmals scannen oder die Nummer von Hand eingeben. Bei einer Netzwerkübertragung können wir die Nachricht einfach nochmals anfragen, falls wir einen Fehler erkennen. Aber z.B. bei einem QR-Code (Abbildung 3) möchten wir den Code auch noch lesen können, wenn einige Teile verloren

gegangen sind. Oder bei einer CD möchten wir die Musik auch noch hören können, wenn die CD Kratzer hat.

Wir möchten nun zusammen einen einfachen Code entwickeln, um auch Fehler korrigieren zu können. Um den Code einfach zu halten, möchten wir momentan nur ein einzelnes Bit korrigieren können. Die tatsächlichen Codes, die z.B. bei QR-Codes angewandt werden, können viele Bits aufs Mal korrigieren, sind aber dafür auch etwas komplizierter.

Eine sehr einfache Methode ist, die Nachricht einfach dreimal hintereinander zu schreiben. Für jedes Bit nehmen wir dann den Wert, der öfter vorkommt. Wir würden z.B. 01000101 als 010001010100010101000101 codieren.

Übung 5

Hier hat sich jeweils ein Fehler eingeschlichen. Was war die ursprüngliche Nachricht?

- 110010001100100011101000
- 110001011100011111000101
- 101110011111100111111001

Dieses Verfahren ist aber nicht sehr effizient. Wir müssen die dreifache Menge an Bits übertragen bzw. speichern, aber können dennoch nur ein einziges Bit korrigieren. Das geht doch sicher besser?

Übung 6

Entwickle ein Verfahren, mit dem du ein einzelnes falsches Bit erkennen und korrigieren kannst. Versuche, möglichst wenige zusätzliche Bits zu benötigen.

Es kann hilfreich sein, die Bits nicht nur in einer Linie auszulegen.

Übung 7

Was passiert, wenn bei deinem Code zwei Bits falsch sind? Was, wenn drei Bits falsch sind?

Kapitel 4: QR-Codes

Die in der Praxis genutzten fehlerkorrigierenden Codes können Fehler mit sehr vielen Bits erkennen und korrigieren, ohne sehr viele zusätzliche Bits zu benötigen. Ein weit verbreitetes Beispiel dafür sind QR-Codes. Hier einige Beispiele:



Abbildung 4: Ein überklebter QR-Code.



Abbildung 5: Ein überklebter QR-Code.



Abbildung 6: Ein überklebter QR-Code.

Übung 8

Welche der QR-Codes können korrekt gelesen werden?

Übung 9

Welche Unterschiede zwischen den Bildern stellst du fest? Wie unterscheiden sich die QR-Codes?

Wie wir gesehen haben, kann bei einem QR-Code eine relativ grosse Fläche verdeckt sein und wir können ihn trotzdem noch lesen. Bei der Erstellung kann man angeben, wie viel Fehlerkorrektur möglich sein soll. Wenn wir mehr Bits korrigieren möchten, brauchen wir dafür natürlich auch mehr Platz (bzw. mehr Bits), aber die heutigen Codes sind relativ effizient. Beim höchsten Level können bis zu 30% des QR-Codes defekt sein und man kann ihn noch immer lesen.

Kapitel 5: Was haben wir gelernt?

Wir haben uns angeschaut, wie ein Computer Fehler erkennen und korrigieren kann. Dies geschieht z.B. mit Prüfziffern bei Barcodes oder mit einem Paritätsbit bei Binärdaten. Verschiedene Verfahren können mehr oder weniger Fehler erkennen bzw. korrigieren. Um mehr Fehler erkennen zu können, benötigen wir aber auch mehr zusätzlichen Speicher. Die heutigen Verfahren sind dabei aber recht effizient und können mit wenig zusätzlichem Speicher viele Fehler korrigieren.

Kapitel 6: Teste dich selbst

- Nenne ein Beispiel, wo ein fehlererkennender Code verwendet wird.
- Nenne ein Beispiel, wo ein fehlerkorrigierender Code verwendet wird.
- Berechne die Prüfziffer der folgenden Produktnummer: 406146238909
- Berechne das Paritätsbit für den folgenden Bitstring: 011101101010
- Warum sind nicht alle QR-Codes gleich anfällig auf Fehler?

Kapitel 7: Bild-Quellen

- Titelbild: AI
- Abbildung 1: Benjamin Schmid
- Abbildung 2: CC by Thomas Schmidt, <https://commons.wikimedia.org/wiki/File:Wikipedia-Buch-Rueckseite-Barcode-OhneBall.jpg?uselang=de>
- Abbildung 3: AI
- Abbildung 4: AI
- Abbildung 5: AI
- Abbildung 6: AI

Kapitel 8: Lösungen

8.1. Kapitel 2: Barcodes

Übung 1: Ändert man eine der Ziffern, wird die Prüfziffer nicht mehr übereinstimmen.

Übung 2: Die Prüfziffer ist 1.

Übung 3: Ein häufiger Fehler beim manuellen Eintippen ist das Vertauschen von zwei benachbarten Ziffern. Da wir die Ziffern abwechselnd mit 1 und mit 3 multiplizieren, wird dies erkannt.

8.2. Kapitel 3: Paritätsbit

Übung 4: Es darf sich nur ein einziges Bit ändern. Sobald sich ein zweites Bit ändert, erkennen wir den Fehler nicht mehr.

Übung 5:

- 11001000
- 11000101
- 11111001

Übung 6: Die möglichen Lösungen sind vielfältig. Einige Beispiele:

- Die Nachricht mit einem Paritätsbit versehen und dann verdoppeln.
- Die Bits in einem Rechteck auslegen, dann pro Zeile und pro Spalte ein Paritätsbit.
- Variation von Hamming Codes.

Übung 7: Bei zwei falschen Bits wird dies als Fehler erkannt, kann aber nicht mehr korrigiert werden. Bei drei falschen Bits wird zwar ein Fehler erkannt, aber er wird zu einem falschen Wert korrigiert.

8.3. Kapitel 4: QR-Codes

Übung 8: Nur Abbildung 5 kann korrekt gelesen werden. Sowohl Abbildung 4 als auch Abbildung 6 haben zu viel Informationsverlust.

Übung 9: Die QR-Codes in Abbildung 5 und Abbildung 6 sind grösser als in Abbildung 4, d.h. sie haben mehr Bits zur Verfügung für die Fehlerkorrektur. Die von den Stickern verdeckte Fläche unterscheidet sich.